IWT SBO Project 060043

Fuzzy Finite Element Method

Vaagheidsgevoelige Eindige-Elementenmethode

Fuzzy Webtools Website

User Manual

Koen Delaere, Sam Corveleyn

Dries Kimpe, Stefan Vandewalle

Scientific Computing Group, KULeuven

Andrei Bularca, Laszlo Farkas

LMS

Jasper Cerneels

Mechanical Engineering, KULeuven

August 2013

Contents

1 Fuzzy Webtools Website – Summary **3**

1.1 Fuzzy Calculator . . . 3

1.2 Example Problems . . . 4

2 Fuzzy Calculator **6**

2.1 Introduction . . . 6

2.2 Membership Function and Fuzzy Number . . . 6

2.3 Declaring a Fuzzy Number . . . 8

2.4 Operations on Fuzzy Numbers . . . 9

2.5 Functions of Fuzzy Numbers . . . 10

3 Plot section **13**

3.1 Introduction . . . 13

3.2 Visualising Fuzzy Numbers . . . 13

3.3 Warning . . . 15

4 Example Problems **18**

4.1 Introduction . . . 18

4.2 Problem description . . . 20

4.3 Input variables . . . 20

4.4 Output variables . . . 21

4.5 Estimate CPU time . . . 22

4.6 Calculate . . . 23

4.7 Analysis and post-processing . . . 24

4.8 Case study: sensitivity analysis . . . 25

5 Solver Settings **29**

5.1 Fuzzy Analysis Terminology . . . 29

5.2 The Solver Settings box . . . 30

5.3 Fuzzy Optimisation Methods 33
5.3.1 Sampling - Transformation Method (TM) 33
5.3.2 Sampling - Equidistant 34
5.3.3 Gradual alpha-level Decreasing (GaD) 35
5.3.4 Response Surfaces and Kriging 36
5.4 Solver Settings and CPU time 37
5.4.1 The "Estimate CPU time" Button 37
5.4.2 Effect of Parameter Values and Method on CPU time 39
5.5 References 43

Chapter 1

Fuzzy Webtools Website – Summary

The Fuzzy Webtools Website at `https://ffem.mech.kuleuven.be/fuzzy/` is an interactive website where the user can experiment with fuzzy numbers. The user can define fuzzy numbers, apply basic fuzzy arithmetic (standard operations and functions) and visualise the resulting membership functions. The site also offers predefined example problems, simple 'Academic Problems' and more advanced 'Engineering Problems', in order to demonstrate the added value of using fuzzy numbers to represent uncertainties in a realistic engineering setting. The user can select the method used for fuzzy optimisation as well as some of the solver settings. A short overview of the website structure and functionality is presented below. For more details, please refer to the relevant chapter.

1.1 Fuzzy Calculator

The **Calculator** tab offers a complete toolbox where the user can experiment with fuzzy numbers. The calculator is divided in two sections. The left section contains all the variables stored in the calculator memory, referred to as the **workspace**, and gives access to the solver settings. The bottom left section contains the plot window. The section on the right gives a full description of the fuzzy calculator. When an example is loaded, the right section gives a full description of the problem in various steps.

A graphical and command line interface is provided to define and manipulate real and fuzzy numbers. Various standard operations (add, subtract, multiply, divide, power, ...) and standard functions (sin, sqrt, exp, log, ...) are available for fuzzy number arguments. Operations and functions can be grouped using brackets (..(..)..). The syntax is explained in detail in Chapter 2. In this calculator, results are computed using Fuzzy Arithmetic.

Any fuzzy or real variables used as *Input Variables* for an example problem will be added to the workspace automatically. Conversely, the *Output Variables* of any successfully

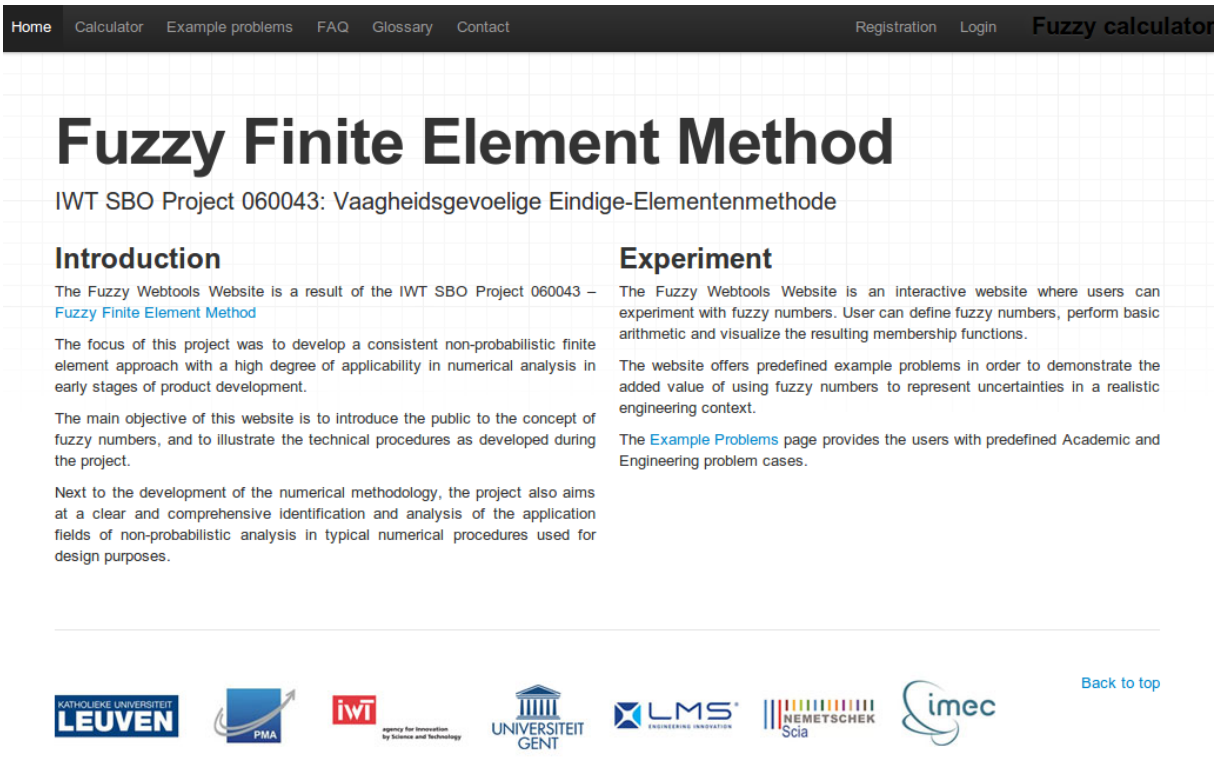
Home
Calculator
Example problems
FAQ
Glossary
Contact
Registration
Login
Fuzzy calculator
Fuzzy Finite Element Method
IWT SBO Project 060043: Vaagheidsgevoelige Eindige-Elementenmethode
Introduction
The Fuzzy Webtools Website is a result of the IWT SBO Project 060043 – Fuzzy Finite Element Method
The focus of this project was to develop a consistent non-probabilistic finite element approach with a high degree of applicability in numerical analysis in early stages of product development.
The main objective of this website is to introduce the public to the concept of fuzzy numbers, and to illustrate the technical procedures as developed during the project.
Next to the development of the numerical methodology, the project also aims at a clear and comprehensive identification and analysis of the application fields of non-probabilistic analysis in typical numerical procedures used for design purposes.
Experiment
The Fuzzy Webtools Website is an interactive website where users can experiment with fuzzy numbers. User can define fuzzy numbers, perform basic arithmetic and visualize the resulting membership functions.
The website offers predefined example problems in order to demonstrate the added value of using fuzzy numbers to represent uncertainties in a realistic engineering context.
The Example Problems page provides the users with predefined Academic and Engineering problem cases.
Back to top

Figure 1.1: Fuzzy Webtools Website (version of August 2013).

completed example problem are automatically added to the workspace.

The **Calculator** tab provides access to the various optimisation methods available for the predefined example problems. Please note that the solver settings do not impact basic operations in the calculator.

In the **Plot** section, the membership functions of one or more fuzzy numbers can be visualised. For more details please refer to Chapter 3.

1.2 Example Problems

On the **Example Problems** tab, the user must select a predefined example problem. The list of example problems is subdivided in "Academic Problems" and "Engineering Problems". The 'Academic Problems' solve ordinary differential equations (ODE) and partial differential equations (PDE), initial value problems (IVP) and boundary value problems (BVP), with fuzzy numbers representing uncertainties on a coefficient (a parameter) or on initial conditions. Some of the 'Engineering Problems' interface with finite element (FE) software to establish a fuzzy finite element analysis (FFE). More information about the example problems is given in Chapter 4.

All example problem results are computed using Fuzzy Optimisation. The actual optimisation method and the relevant solver settings are selected and set in the **Solver Settings** box on the **Calculator** tab . More details on methods and settings are given in Chapter 5.

Chapter 2

Fuzzy Calculator

2.1 Introduction

The **Fuzzy Calculator** tab offers a graphical and command line interface to define and manipulate real and fuzzy numbers. Various standard operations (add, subtract, multiply, divide, power, ...) and standard functions (sin, sqrt, exp, log, ...) are available for fuzzy number arguments. Operations and functions can be grouped using brackets (..(..)..). In this calculator, results are computed using Fuzzy Arithmetic.

All variables are stored in the calculator memory, referred to as the **workspace**.

Any fuzzy or real variables used as *Input Variables* for an example problem will be added to the workspace automatically. Conversely, the *Output Variables* of any successfully completed example problem are automatically added to the workspace.

All fuzzy calculator results are computed using Fuzzy Arithmetic.

2.2 Membership Function and Fuzzy Number

Figure 2.1 shows four examples of a **membership function**: triangular, trapezoidal, interval and general, user-defined shape. The value $\mu(x)$ of the membership function covers the interval [0,1]. The interval $[a_1,b_1]$ for which $\mu(x)=0$ is called the *support*. The interval $[a_N,b_N]$ for which $\mu(x)=1$ is called the *top*. Between support and top a number of other levels or intervals $[a_i,b_i]$ can be specified. These levels are called *alpha-levels* or *alpha-cuts*. In general the membership function is represented by a set of N intervals $[a_1,b_1:L_1;\ a_2,b_2:L_2;\ ...;\ a_N,b_N:L_N]$ where $L_1=0$, $L_N=1$ and $N \geq 2$. Note that the four examples of Figure 2.1 have different numbers of alpha-levels.

A membership function is called **convex** if 1) every alpha-level is contained in the ones

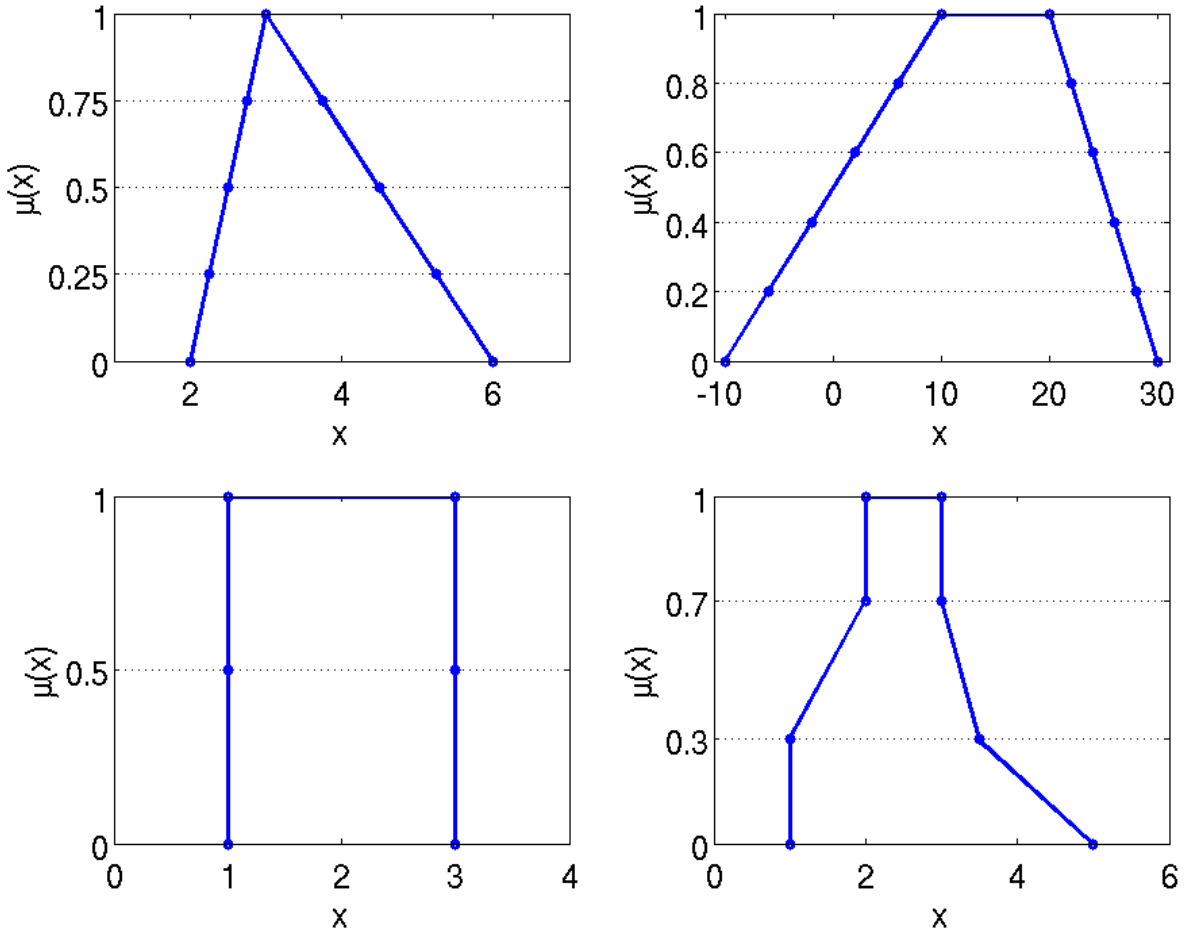

Figure 2.1: Example membership functions: triangular (top left), trapezoidal (top right), interval (bottom left) and with general, user-defined shape (bottom right). Note also that the four examples have different numbers of alpha-levels.

below it: $[a_j,b_j] \in [a_i,b_i]$ for all $j > i$, and 2) every alpha-level $[a_i,b_i]$ is itself convex in the classical sense.

A **fuzzy number** is defined by a membership function that is convex. The value of $\mu(x)$ can be interpreted as the degree of membership of the value x in a fuzzy set. This degree of membership can be used to represent an uncertainty on e.g. a material parameter, geometrical dimension, etc.

A real number r can be represented by a *crisp* membership function, where the support, the top, and every other alpha-level are all equal to $[r,r]$.

2.3 Declaring a Fuzzy Number

The fuzzy calculator provides various predefined shapes to define a membership function: triangular, trapezoidal, interval, gaussian and lognormal. There is also the possibility to define other, general shapes. Table 2.1 gives an overview.

syntax	*description*
triag (left, center, right)	constructs triangular fuzzy number
trap (left, top_{left}, top_{right}, right)	constructs trapezoidal fuzzy number
interval (left, right)	constructs interval fuzzy number: top=support
gauss (mean, sigma)	constructs gaussian fuzzy number – arguments are the mean and standard deviation of the variable
lognormal (mean, sigma)	constructs log-normal fuzzy number – arguments are the mean and standard deviation of the natural logarithm of the variable
[a_1,b_1:L_1; a_2,b_2:L_2; ...; a_N,b_N:L_N]	constructs fuzzy number by enumerating the alpha-levels [a_i,b_i] from support (L_1=0) to top (L_N=1).
discretize (`f`, N)	resamples fuzzy number `f` at N different alpha-levels

Table 2.1: Membership function definition.

On the left section of the calculator, we find the **Variables** box. Toggle this box to create a new variable. To define the triangular fuzzy number shown at the top left in Figure 2.1, choose a name and a type. Next, provide the content of the new variable with it's respective syntax. Alternatively, toggle the response button of the Calculator box. To define variables using the command line interface, type `a = triag(2,3,6)` in the fuzzy calculator input text field, and press enter:

Input expression and press enter : `a = triag(2,3,6)`

The calculator output text area repeats your input text and the parser's interpretation of it. The fuzzy variable `a` is now stored in the calculator memory, or workspace, and is available in the Variables box of the graphical interface. To define the other membership functions of Figure 2.1, type the following declarations:

```
b = trap(-10,10,20,30)
c = interval(1,3)
d = [1,5:0; 1,3.5:0.3; 2,3:0.7; 2,3:1]
```

We used the general format [a_1,b_1:L_1; a_2,b_2:L_2; ...; a_N,b_N:L_N] to define fuzzy number `d` because its membership function does not have a predefined shape. There are various ways to define a crisp membership function, for example:

`r = interval(5,5)` or

`r = [5,5:0; 5,5:1]` or

`r = triag(5,5,5)` etc.

The default number of alpha-levels N applied when declaring fuzzy numbers is 10. To adjust the number of alpha-levels in the membership function, use the `discretize` function (see the last row in Table 2.1):

`a = discretize(a,5)`

This will redefine the fuzzy number `a` from 10 alpha-levels (default) to the 5 levels shown in Figure 2.1. In general, the command

`f = discretize(g,n)`

will define a new fuzzy number `f` which is equal to an existing fuzzy number `g` but represented by `n` alpha-levels. To discretize a variable using the graphical interface, create a new variable of the type `Discrete fuzzy number`, or edit the variable to change the type, providing the variable name and desired number of alpha-levels as arguments.

To define two fuzzy numbers `a` and `b` corresponding to the triangular and trapezoidal membership functions of Figure 2.1, type the four commands shown in Figure 2.2:

Note that after the `discretize` command, the calculator reverts to the general format [a_1,b_1:L_1; ...; a_N,b_N:L_N]. This is always the case when operations are performed on fuzzy numbers.

2.4 Operations on Fuzzy Numbers

Table 2.2 lists standard operations that the calculator can perform on fuzzy numbers. Some examples are shown in Figure 2.3. The results of the operations can be assigned to new fuzzy numbers `c` and `d`. Use brackets to group operations.

Note that division by zero is not allowed. For fuzzy numbers this means that `a/b` is not allowed because the support of `b` contains 0. Note also that the power operation `b^a` is not allowed because the result contains complex numbers (due to raising the negative numbers contained in the support of `b` to a non-integer power).

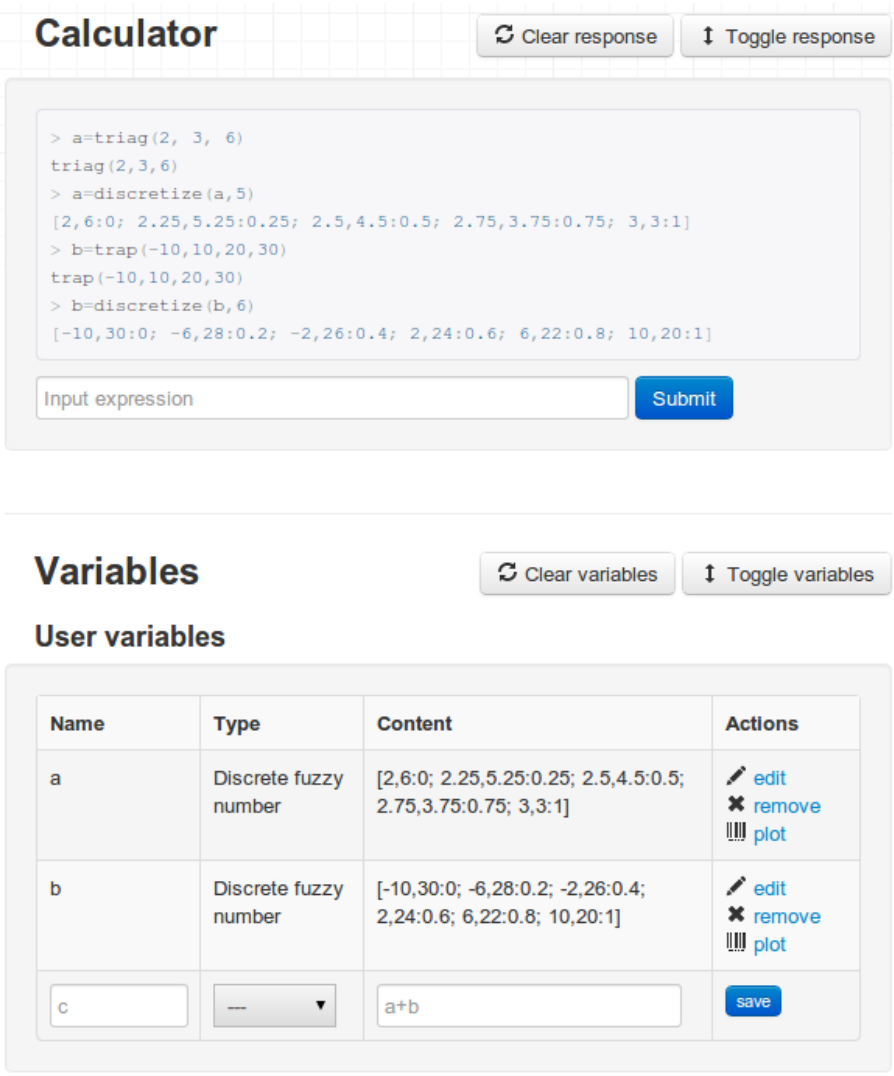

Figure 2.2: Define fuzzy numbers `a` and `b`.

2.5 Functions of Fuzzy Numbers

Table 2.3 lists standard functions that the calculator can perform on fuzzy numbers. Some examples are shown in Figure 2.4. The results of the operations can be assigned to new fuzzy numbers `c` and `d`.

Note that taking the square root of a negative number is not allowed. For fuzzy numbers this means that `sqrt(b)` is not allowed because the support of `b` contains negative numbers. Similar standard restrictions are in place for the other functions.

Figure 2.3: Fuzzy number operations.

syntax	*operation*
a + b	addition
a – b	subtraction
–a	unary minus
+a	unary plus
a * b	multiplication
a / b	division
a ∧ b	power

Table 2.2: Fuzzy number operations available in the calculator.

Figure 2.4: Fuzzy number functions.

syntax	*function description*
abs(a)	absolute value
exp(a)	exponential function
log(a)	natural logarithm
sqrt(a)	square root
sin(a)	sine
cos(a)	cosine
tan(a)	tangent
asin(a)	inverse sine
acos(a)	inverse cosine
atan(a)	inverse tangent

Table 2.3: Fuzzy number functions available in the calculator.

Chapter 3

Plot section

3.1 Introduction

Membership functions of fuzzy numbers can be visualised in the plot section available at the **Calculator** tab. The *Variables* box contains all the variables available in the workspace. Each variable has a plot and edit button. Clicking the plot button visualises the membership function currently assigned to that variable. Alternatively, variables can be plotted by entering the name directly into the input line of the plot section.

3.2 Visualising Fuzzy Numbers

Let's again define fuzzy numbers `a` and `b` corresponding to the triangular and trapezoidal membership functions of Figure 2.1. At the fuzzy calculator command line, type the four commands shown in Figure 2.2 or create the variables using the graphical interface provided by the *Variables* box.

To visualise the membership function of fuzzy number `a`, click the plot button of that variable. Alternatively, type "a" on the input text line of the plot section and click the **plot** button, or press enter.

By clicking the plot button of multiple variables, the membership functions can be plotted simultaneously. To visualise two or more membership functions via the input text line, separate their name or expression with a vertical bar. For example, to display both fuzzy numbers `a` and `b`, type "a | b" in the input text line and press enter. Both membership functions appear on the plot window, using different colors. A color legend is shown below the plot window.

It is also possible to type a set of operations and/or functions into the input text line,

for example "b/a". Note however that an assignment such as "c = b/a" is *not* allowed in the plot window input line.

Figure 3.1 demonstrates the plotting functionality discussed above, by visualising simultaneously the membership function of fuzzy number `a` (yellow), fuzzy number `b` (blue), the division of fuzzy numbers `b/a` (red) and the crisp membership function for the number 18 (green) :

Enter expression : `a | b | b/a | 18`

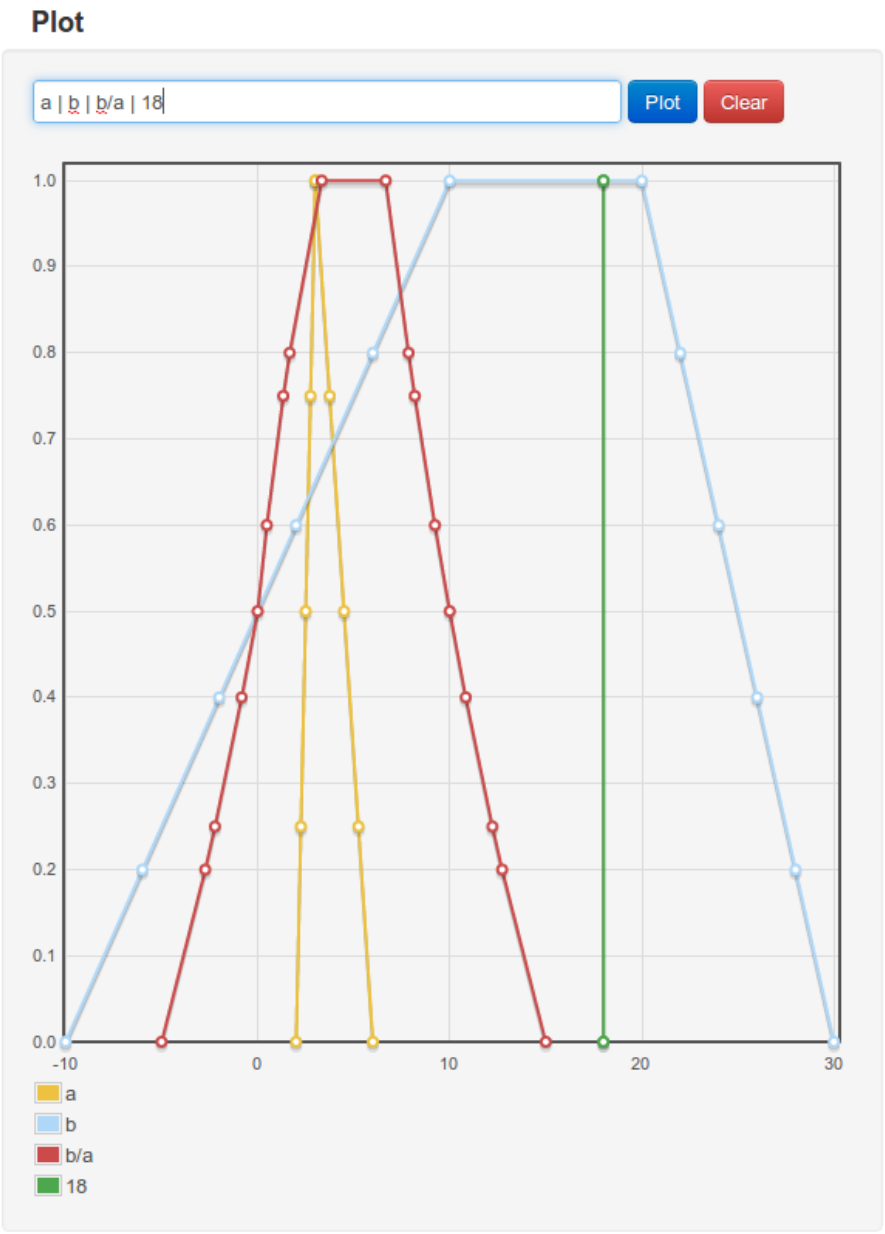

Figure 3.1: Use the plot window to visualise multiple membership functions.

Note that fuzzy numbers `a` and `b` do *not* have the same number of alpha-levels. The set of alpha-levels of the result of the division `b/a` is the *union* of the sets of alpha-levels of `a` and `b`. This can be seen clearly on Figure 3.1, where all alpha-levels are indicated with round bullets.

When the mouse is hovering over the plot window, a yellow text box appears saying *Click a point to display exact coordinates*, see Figure 3.2. Use the mouse to click on any round bullet and a message window pops up with the exact numerical values of the coordinates $(x, \mu(x))$ of the point selected, see Figure 3.3.

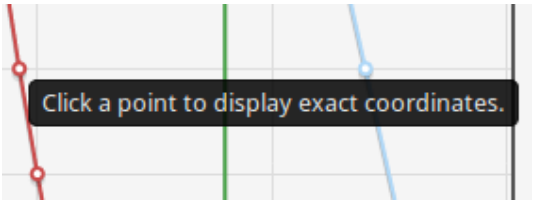

Figure 3.2: This message appears when the mouse is hovering over the plot window.

3.3 Warning

The legend of the plot window gives a **Warning** in case the result obtained using Fuzzy Arithmetic may be too conservative. In that case Fuzzy Optimisation may give a better (less conservative) result. Currently, the fuzzy calculator uses Fuzzy Arithmetic only, while the example problems are solved using Fuzzy Optimisation.

The warning can be seen in Figure 3.4, where `b*b` draws the warning while `b^2` does not. Comparing the membership functions for both expressions, it can be seen that the result for `b*b` (yellow) is indeed too conservative.

In Fuzzy Arithmetic, the expression $b * b$ is interpreted as $b_1 * b_2$ where the two factors b_1 and b_2 are instanced independently, even as they refer to the same fuzzy number. Therefore the result will be different from $b \wedge 2$ which contains only one factor.

In Fuzzy Optimisation, factors that refer to the same fuzzy number are *not* instanced independently and thus the result for $b * b$ and $b \wedge 2$ is the same.

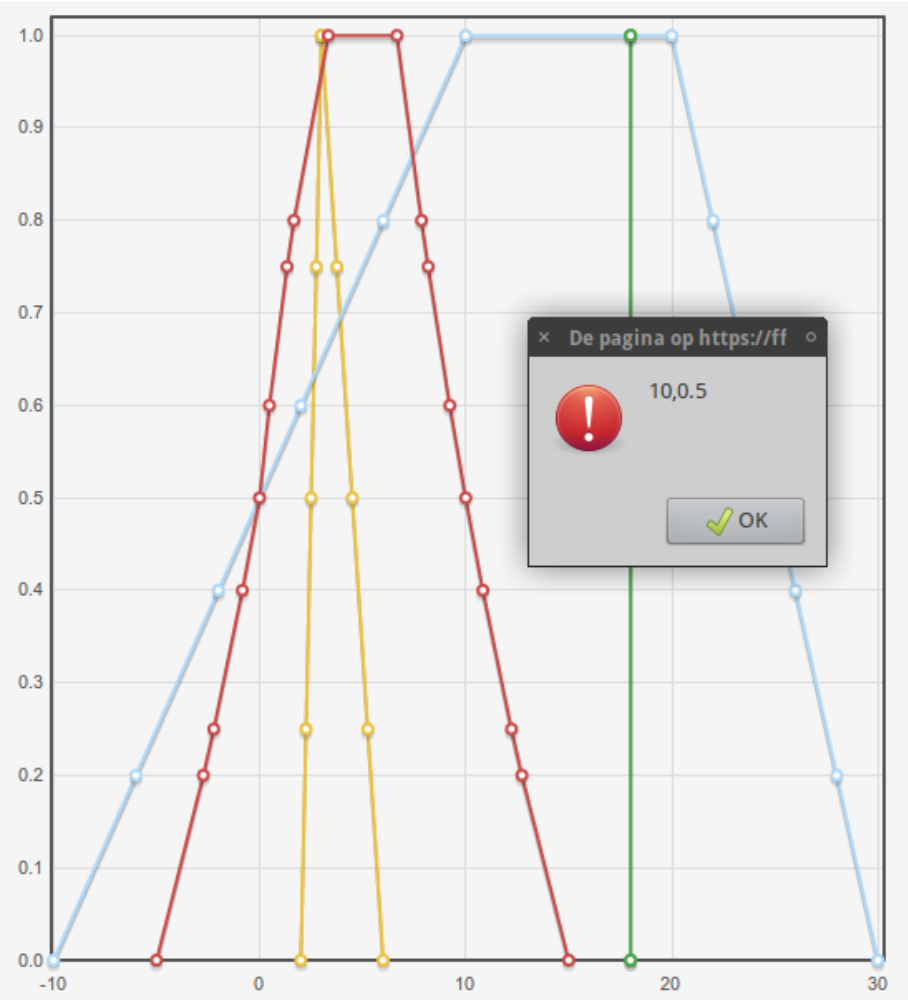

Figure 3.3: Click one of the round bullet points on the plot window to probe the exact coordinates of any membership function vertex.

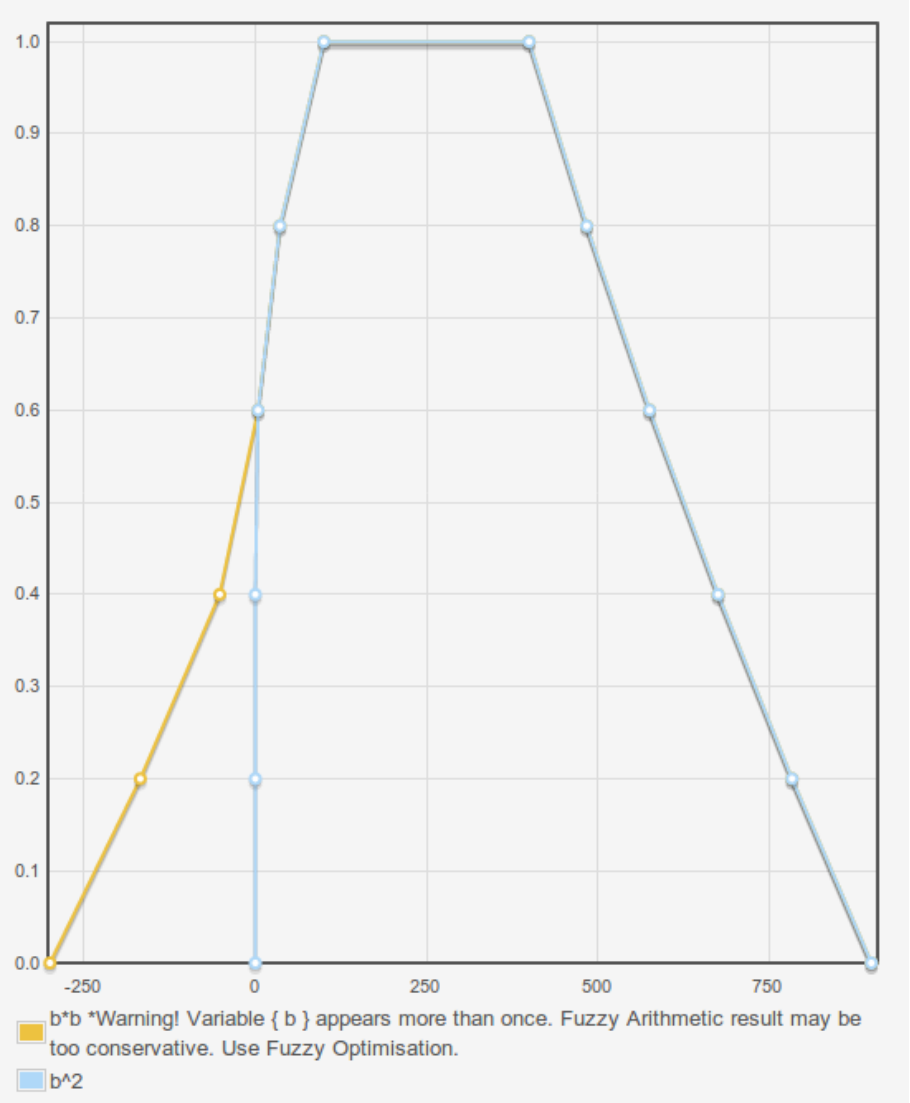

Figure 3.4: The plot window legend displays a warning in case the result may be too conservative.

Chapter 4

Example Problems

4.1 Introduction

Click on the **Example Problems** tab. The user is prompted to select a problem from the list, as in Figure 4.1.

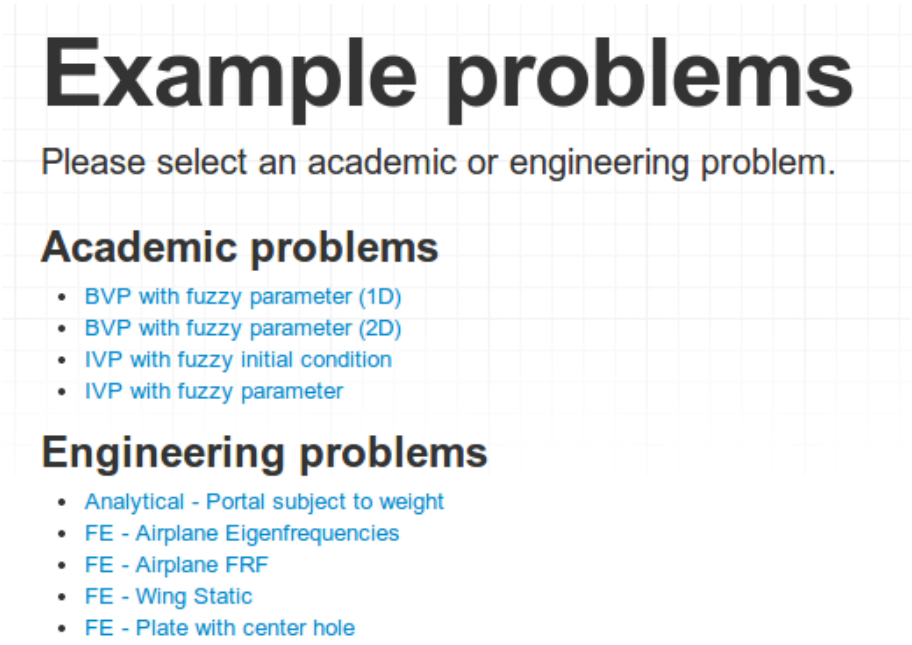
Example problems
Please select an academic or engineering problem.
Academic problems
BVP with fuzzy parameter (1D)
BVP with fuzzy parameter (2D)
IVP with fuzzy initial condition
IVP with fuzzy parameter
Engineering problems
Analytical - Portal subject to weight
FE - Airplane Eigenfrequencies
FE - Airplane FRF
FE - Wing Static
FE - Plate with center hole

Figure 4.1: Example Problems list.

The Academic Problems solve ordinary differential equations (ODE) and partial differential equations (PDE), initial value problems (IVP) and boundary value problems (BVP), with fuzzy numbers representing uncertainties on coefficients (parameters) or on initial conditions.

The Engineering Problems aim at solving more complex and realistic engineering problems. Some of the engineering problems interface with finite element (FE) software to establish a fuzzy finite element analysis (FFE).

All example problem results are computed using Fuzzy Optimisation. The actual optimisation method and the relevant solver settings are selected and set in the **Solver Settings** box. More details on methods and solver settings are given in Chapter 5.

IVP with fuzzy parameter

We compute the solution of an **Initial Value Problem** (IVP) with fuzzy parameters, using a toolbox for fuzzy optimization developed at the KULeuven. The differential equation we are solving represents a mass-spring-damper system:

$$m\ddot{y} + c\dot{y} + ky = 0$$

where y(t) is the position of the mass, m is its weight, k is the spring constant and c is the damping coefficient. In this example, the damping coefficient c and the spring constant k are not regular real numbers with a single value, but they are fuzzy numbers defined by a membership function.

The default membership function of c is shown in Figure 1 below. The user is free to define different membership functions for c and k using the input fields under **Input variables**. It is possible to specify a real value for a fuzzy input parameter. The user can also choose mass m, start time t0, end time tfinal, and initial position y0 and initial velocity v0 of the mass. Push the **Calculate** button to solve the differential equation using fuzzy optimization.

The resulting fuzzy solution at final time y(tfinal) is stored in the **Output variable** yfinal and written to the *Fuzzy Calculator* and *Plot* workspace. A 2D plot of the total fuzzy solution y(t) is written to file.

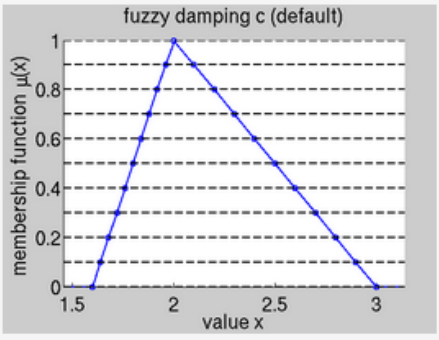

Figure 1. The default triangular membership function of damping coefficient c.

Figure 4.2: IVP with fuzzy parameter: problem description.

Every problem has the same structure and solving a problem always follows the same scenario. The various steps are illustrated here by example of one of the academic problems.

Select the IVP with fuzzy parameter problem, see Figure 4.2.

4.2 Problem description

After selecting a problem, the top of the window shows the problem title and problem description. The text describes the problem to be solved and its parameters, using relevant equations and diagrams. Pay attention to which uncertainties are represented by fuzzy numbers.

Example – For our example IVP with fuzzy parameter, the problem consists of solving the ordinary differential equation (ODE) that represents an unforced mass-spring-damper system. The initial position y_0 and initial velocity v_0 of the mass are taken into account so this constitutes an initial value problem (IVP). The spring stiffness k and damping coefficient c are fuzzy numbers, while the mass m and all other parameters are a real numbers. The challenge is thus to solve an ODE with fuzzy coefficients.

4.3 Input variables

Input variables

Name	Type	Content	Actions
c	Triangular fuzzy number	triag(1.6,2,3)	edit plot
k	Triangular fuzzy number	triag(45,50,55)	edit plot
m	Integer	1	edit plot
t0	Integer	0	edit plot
tfinal	Integer	2	edit plot
v0	Integer	5	edit plot
y0	Integer	1	edit plot

Figure 4.3: IVP with fuzzy parameter: Input Variables.

The selected example problem is loaded into the Fuzzy Calculator. In the left section, we find the Input variables box (see Figure 4.3) which lists all real-valued and fuzzy problem parameters by symbol (m), input text field with default value (1), description (mass) and unit (kg). Click the plot button of a variable to view the membership function currently assigned to that fuzzy input variable (see Figure 4.3).

The user is free to change the default value of any input variable by typing a value, expression or workspace variable name into the text field. For a **real-valued** input variable, the text field can contain

- a real number, e.g. 2,
- the name of a real-valued variable present in the workspace, e.g. m2,
- an expression that resolves to a real number, e.g. 0.2*10 or sqrt(m2/5),

where we assumed that m2 represents some positive, real number. For a **fuzzy** input variable, the text field can contain

- anything that is allowed for a real-valued input variable (see list above),
- the name of a fuzzy-valued variable present in the workspace, e.g. a,
- an expression that resolves to a fuzzy value, e.g. 0.2*a or sqrt(a/5),

where we assumed that `a = triag(2,3,6)` as in previous chapters.

Note that it is possible to assign a real number (crisp membership function) to a fuzzy input variable, while it is not possible to assign a membership function to a real-valued input variable. If the user assigns real number values to all fuzzy input parameters, then only one single deterministic run will be performed.

Example – For our example IVP with fuzzy parameter, the input variable c is a fuzzy number in order to represent real-life uncertainty on the damping coefficient. The default value for c is the triangular membership function `triag(1.6,2,3)`. The user is free to specify alternative real or fuzzy values for c and k. Apart from c and k, the other (real-valued) input variables can be freely adjusted as well, e.g. the initial position y_0 and initial velocity v_0 of the mass.

4.4 Output variables

Below the Input variables box we find the Output variables box (see Figure 4.4). This box lists all problem outputs (results). A short description of the output variable is available

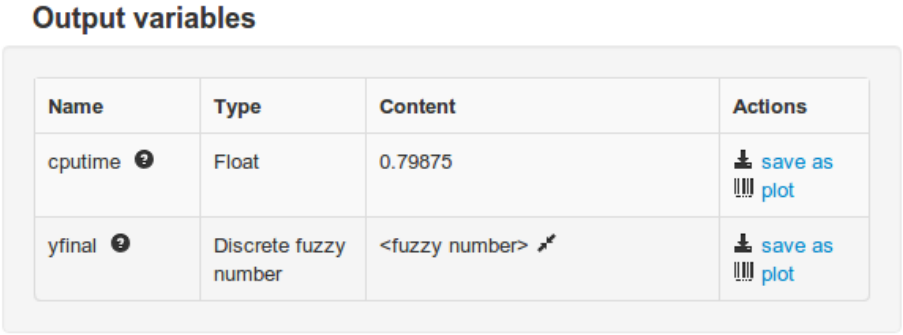

Figure 4.4: IVP with fuzzy parameter: Output Variables.

by hovering over the question mark. After successful completion of an example problem the output variables are automatically added to the calculator workspace, using the pre-defined names shown in the text field. Output variables can be saved under a new name, this allows you to store results from different analyses under different names, and thus avoid overwriting previous results. All saved variables are stored into the User Variables box.

Example – For our example IVP with fuzzy parameter, the first output variable y_{final} captures the membership function for the final position of the mass, i.e. the position of the mass at time $t = t_{final}$. Solve the problem with mass $m = 1$ using default solver settings. Save the result y_{final} in the calculator workspace under the name `yfinal1`. Now change the value for mass so that $m = 2$ and solve the problem a second time. Save the resulting y_{final} as `yfinal2`. Go to the **Plot** section to compare the membership functions of the two results for $m = 1$ and $m = 2$:

```
Enter expression:  yfinal1 | yfinal2
```

If the user does not save the results under a new name, then the result of the second $m = 2$ run will overwrite the result of the first $m = 1$ run because only one single variable name `yfinal` is used.

4.5 Estimate CPU time

The user should click the Estimate CPU time button before the calculation of the problem is started. This is done in order to inform the user of the expected total CPU time needed, and to warn the user in case the chosen solver settings are such that they lead to an excessively large CPU time. The maximum CPU time is limited to 30 minutes. When a user clicks the calculate button,

After clicking the Estimate CPU time button, several lines of information appear, for

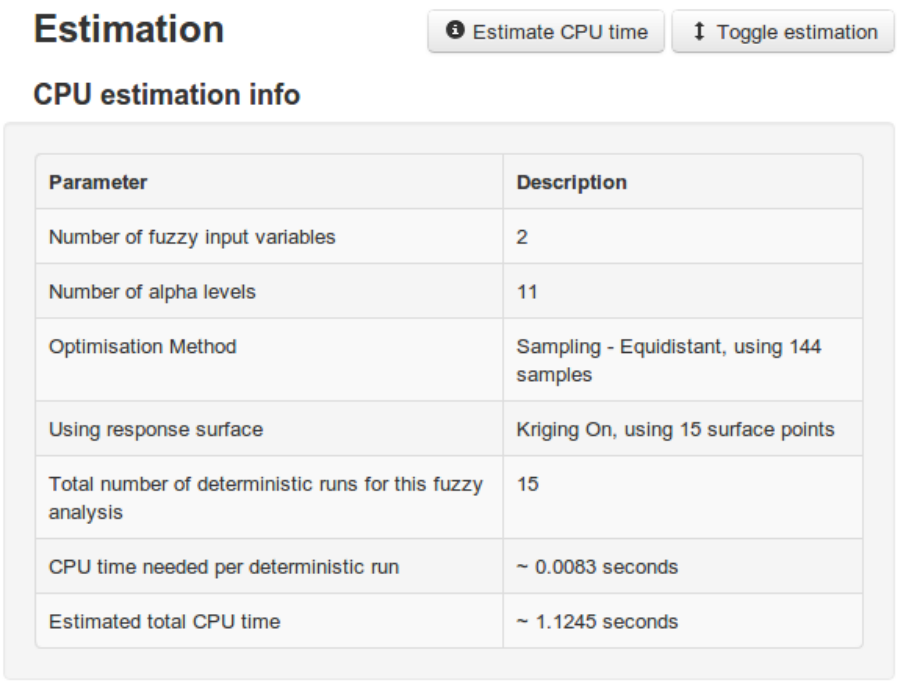

Figure 4.5: Information given after clicking the Estimate CPU time button.

example as in Figure 4.5. The upper lines will be discussed in Chapter 5, Solver Settings, Section 5.4.1. Here we only take note of the last three lines:

- Total number of deterministic runs: the number of times the deterministic core module will be run in order to complete the fuzzy analysis,
- CPU time needed per deterministic run: CPU time needed for one single run of the deterministic core module of the selected problem,
- Estimated total CPU time: the product of the previous two lines.

Most example problems finish in one minute or less when using the default solver settings. If the last line Estimated total CPU time indicates more than 30 minutes then the user must consider selecting different solver settings and. If the user intends to wait for the results of the fuzzy analysis, then the estimated total CPU time is a good indication of how much time is needed to complete the fuzzy analysis. **Do not close the browser while a fuzzy analysis is running, or the results will be lost**.

4.6 Calculate

After clicking the Calculate button, a dialog window appears informing the user of the total CPU time. The user must accept in order to calculate the problem, or press cancel

to change the solver settings. During analysis, a Calculating... box is shown in the middle of the screen, and navigation on this website is turned off by means of a dark grey overlay. The user must wait until the fuzzy analysis is finished before the overlay disappears and navigation is possible again. Do not close the browser while waiting, or the results will be lost.

When the fuzzy analysis is finished, the area below the Calculate button contains some information, with headers Problem Input and Problem Output. This is a good place to check for error messages in case something goes wrong. In particular the error message `Machine too busy` will appear here if the server load is too high.

Example – For our example IVP with fuzzy parameter, the deterministic core module consists of solving one differential equation with real number values assigned to all coefficients. It is the fuzzy analysis module (Fuzzy Optimisation library) that manages the results of various deterministic runs and assembles the membership function of the output variables corresponding to the membership functions of the fuzzy input variables.

4.7 Analysis and post-processing

Upon successful completion of the fuzzy analysis, a section Analysis appears which contains automatically generated, pre-defined figures and text presenting certain output variables and results, see Figure 4.6.

The output variables are now also available in the fuzzy calculator workspace. Go to the **Variables** box to probe these variables, use them in expressions or do some other kind of post-processing. Go to the **Plot** section to plot the results i.e. visualise the membership functions of any output variable.

Example – For our example IVP with fuzzy parameter, the pre-defined figure displays the membership function for the position of the mass as a function of time, $y(t)$, for times $t_0 < t < t_{final}$. The color code is such that the black part indicates the top of the membership function of $y(t)$ and the lighter colors indicate the sides of the membership function. Due to the uncertainty on the damping coefficient c and spring stiffness k, the exact position of the mass becomes more and more fuzzy as time goes on. In this example however, the blurring effect of the uncertainties on c and k is counter-acted by the fact that the mass must eventually end up at its rest position $y(t = \infty) = 0$.

Analysis

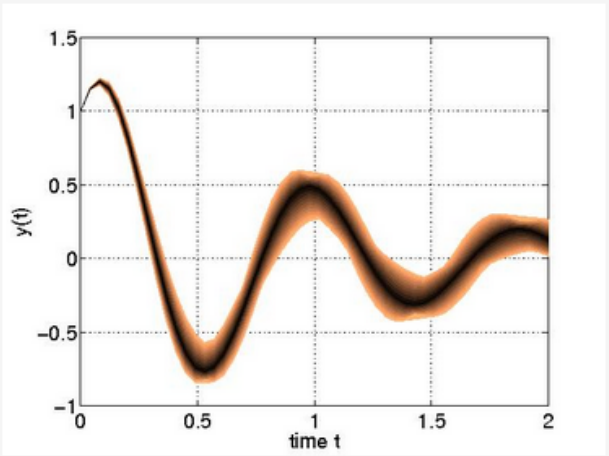

Figure y-t-solution :
Mass-spring-damper system: fuzzy solution for position y with fuzzy damping parameter c.

The uncertainty on y(t), the position of the mass, increases in the presence of uncertain parameters, in the default case these are the fuzzy damping coefficient c and fuzzy spring constant k. This is an example of a situation where an uncertainty becomes more and more important as the system evolves. At what value of c do we have critical damping? What happens when the membership function of c covers both the underdamped and overdamped regime?

Figure 4.6: Automatically generated figures and text in the Analysis section.

4.8 Case study: sensitivity analysis

We continue to use the academic example problem IVP with fuzzy parameter to demonstrate how to perform a simple sensitivity analysis using fuzzy numbers.

Go to the Input variables section and set the input variable k (spring stiffness) to a real value (say $k = 50$) so that all fuzziness is due to the fuzzy input parameter c, the uncertain damping coefficient of the system.

Keep the default value $c =$ `triag(1.6,2,3)` and solve the problem by pressing the Calculate button. The resulting figure under Analysis shows the fuzzy position of the mass as a function of time, see Figure 4.7. This result is different from Figure 4.6 because the spring stiffness k is no longer fuzzy.

Figure 4.7 demonstrates that for this set of input values, the mass-spring-damper system is underdamped, and the mass has not yet come to rest after two seconds ($t = 2$s). We would like to see the mass settle at its rest position (or close to its rest position) in a shorter time, and so we investigate higher values of the damping coefficient c. Go back to the Input variables section and type `triag(1.6,12,13)` in the input text field for c. Press the Calculate

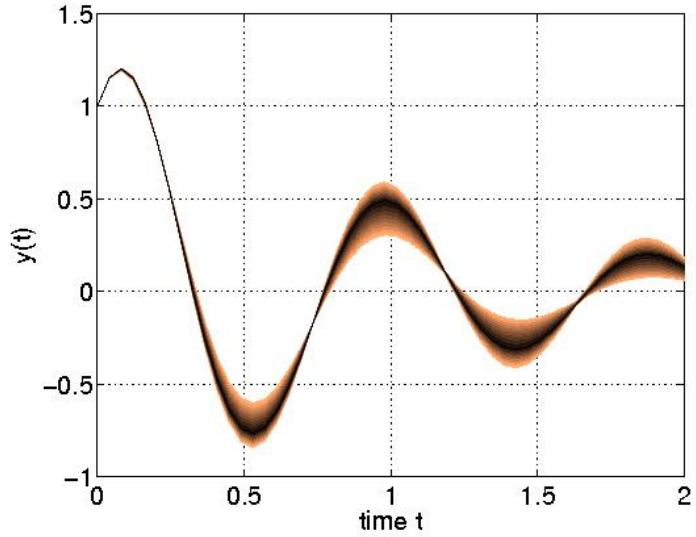

Figure 4.7: IVP with fuzzy parameter result: position of mass $y(t)$ for c =triag(1.6,2,3) and $k = 50$. The system is underdamped.

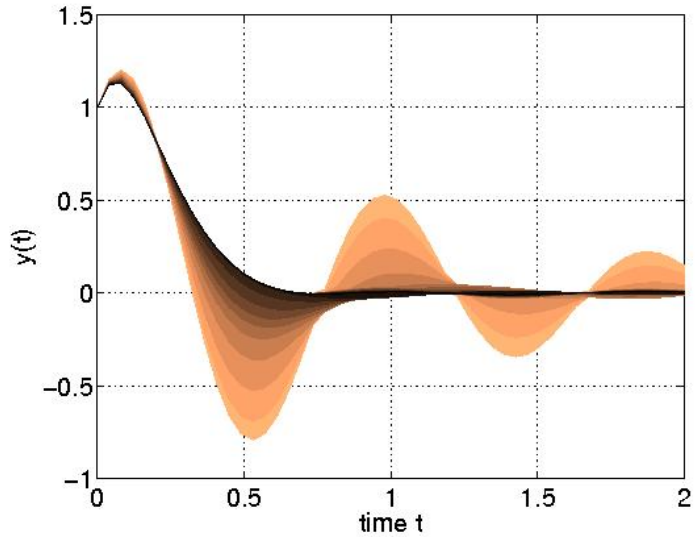

Figure 4.8: IVP with fuzzy parameter result: position of mass $y(t)$ for c =triag(1.6,12,13) and $k = 50$. On some alpha-levels, the system is critically damped and overdamped.

button to solve the problem once again. For clarity, go to the Output variables box and save yfinal as `yfinal_crit`.

The resulting mass position $y(t)$ using damping $c =$ `triag(1.6,12,13)` is shown in Figure 4.8. The lowest alpha-levels (lightest colors) still show some oscillation and are underdamped. But the higher alpha-levels (darker colors) clearly show critically damped and overdamped motion.

Go to the **Plot** section and display the membership function for `yfinal_crit`, see Figure 4.9. For clarity, the user can display the zero mark by adding the crisp membership function for the number 0:

```
Enter expression:  0 | yfinal_crit
```

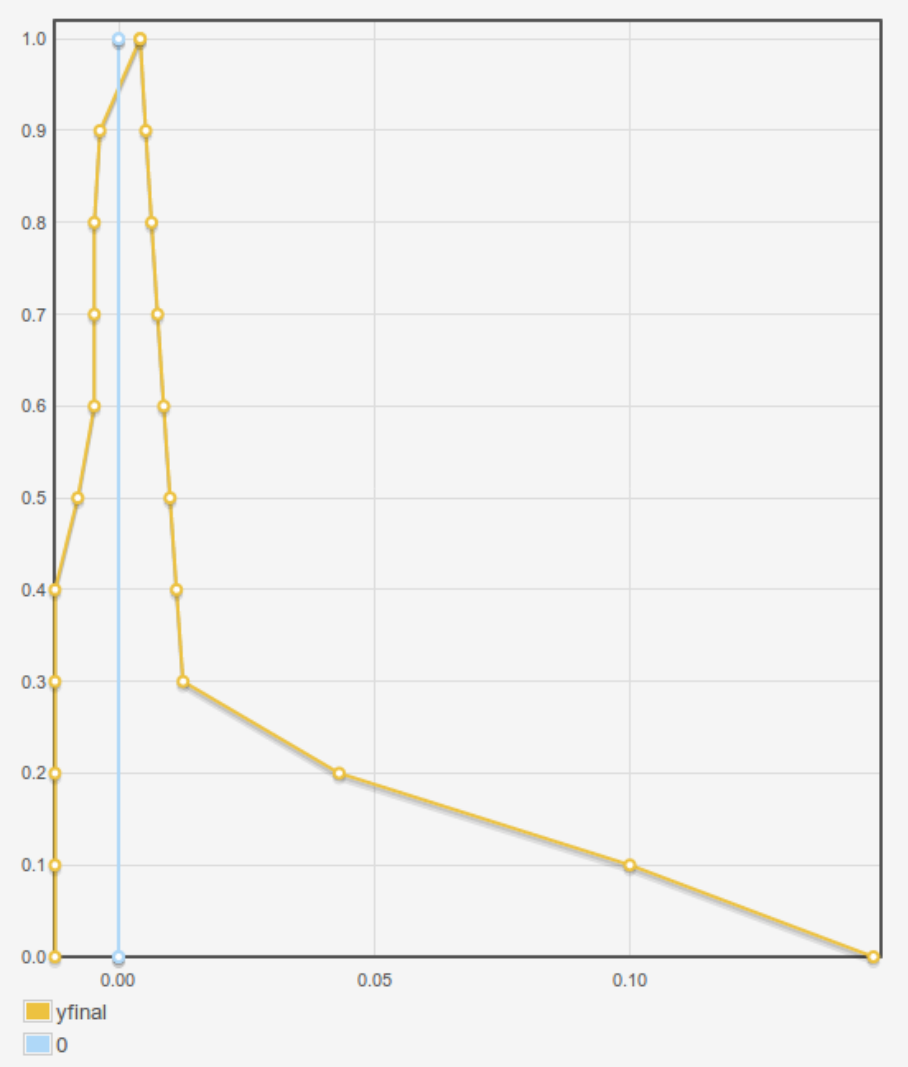

Figure 4.9: IVP with fuzzy parameter result: `yfinal_crit` is the position of the mass at time $t_{final} = 2$s for $c =$triag(1.6,12,13) and $k = 50$. The vertical blue line indicates zero.

The yellow line shows the membership function for $y_{final,crit}$, i.e. the position of the mass at time $t_{final} = 2$s using the higher damping values $c =$ `triag(1.6,12,13)`. Upon investigation of the membership function, we see that for alpha-levels 1.0 down to 0.3, the mass is less than 2cm removed from its rest position, while for alpha-levels below 0.3, the mass is not guaranteed to be that close to its rest position.

With this threshold of "alpha-level 0.3" we now return to the membership function of the fuzzy damping coefficient c. In the plot window, visualise c by typing

```
Enter expression:  triag(1.6,12,13)
```

The triangular membership function of c cuts alpha-level 0.3 at coordinates (4.72 ; 0.3) on the left slope, and (12.7 ; 0.3) on the right slope. This means that a damping coefficient between 4.72 and 12.7 is adequate to bring the mass closer than 2cm from its rest position within two seconds.

Chapter 5

Solver Settings

5.1 Fuzzy Analysis Terminology

For clarity, this section informally introduces some relevant terminology that is used throughout this chapter to discuss the meaning and effect of various solver settings used for the fuzzy optimisation.

At the heart of the fuzzy analysis, we find the so-called **deterministic core**. This is the basic engineering problem or equation, solved using real-valued inputs and parameters, and delivering real-valued outputs. Solving such a determined problem once is called a **deterministic run**.

When the real-valued inputs are replaced by fuzzy numbers (membership functions), how do we obtain the membership functions of the fuzzy outputs? In the *alpha-level strategy* of **Fuzzy Analysis**, an *interval analysis* is performed on each alpha-level α_i of the membership functions of the inputs. Given that all inputs have values contained within their respective α_i-level interval, the interval analysis determines the interval of all possible values for each output, thus defining the α_i-level interval of the membership functions of the outputs.

The entire fuzzy analysis therefore requires a number of deterministic runs, in order to represent the various α_i-level intervals of the fuzzy inputs. All output values of all deterministic runs are assembled into the α_i-level intervals of the fuzzy outputs, as illustrated in Figure 5.1. The number of fuzzy input variables F determines the **dimension** of the fuzzy analysis.

Since every higher alpha-level is contained within the lower alpha-levels (by convexity of the membership function, see Section 2.2), it is possible to organise the fuzzy analysis so that the amount of flops or cpu time is minimized and/or the amount of information that is reused is maximized. Moreover, performing the interval analysis on the lower alpha-level α_i, given the results of the already-finished interval analysis on a higher alpha-level α_j

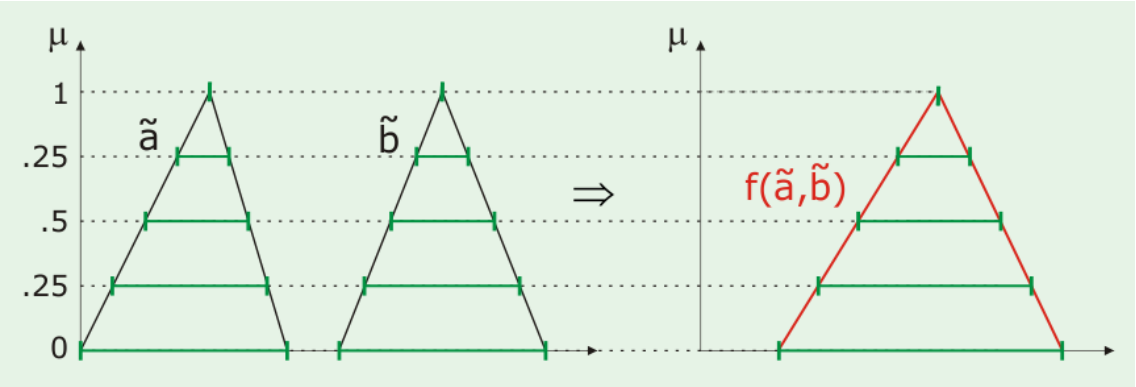

Figure 5.1: Fuzzy Analysis, alpha-level strategy: perform an interval analysis on each alpha-level, and combine results to form the membership function of the output.

can be interpreted as an optimisation problem. Hence the various strategies of organizing an optimal fuzzy analysis are referred to as **Fuzzy Optimisation** and **optimisation method**.

The number of deterministic runs required to complete the fuzzy analysis, and for which sequence of input values the deterministic core is solved, depends on the optimisation method chosen by the user. Getting the real-valued outputs for one particular choice of real-valued inputs is called a **sample**. This sampling is preferably performed on the deterministic core itself (for accuracy), but it can also be performed on a response surface built specifically to represent the deterministic core (for speed).

The F-dimensional **response surface** is built beforehand using a limited number of deterministic runs. Note that this takes up a small amount of cpu time as well. All subsequent samples for the fuzzy analysis are then performed directly on this response surface rather than on the deterministic core itself. Sampling the response surface is usually computationally much cheaper, especially when a deterministic run involves a finite element analysis. One specific technique of building a response surface is called **Kriging** and is available on the fuzzy webtools website.

5.2 The Solver Settings box

Figure 5.2 presents an overview of the **Solver Settings** box, which allows the user to select the optimisation method used for the fuzzy analysis, as well as the main parameters for each method. The user also has the option to build a response surface (Kriging). The Solver Settings box consists of six fields separated by horizontal lines:

Number of Alpha Levels : Specify the number of alpha-levels N on which the fuzzy analysis will be performed. N must be an integer between 2 and 36. The default value

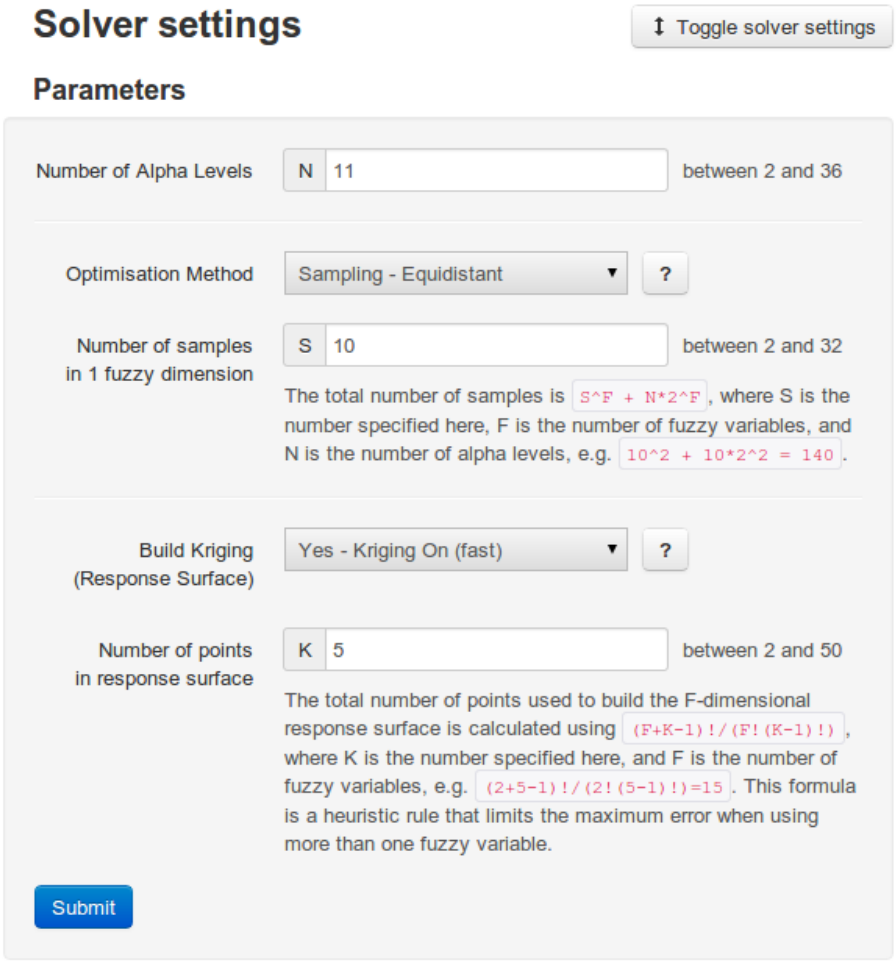
Solver settings
Toggle solver settings
Parameters
Number of Alpha Levels
N
11
between 2 and 36
Optimisation Method
Sampling - Equidistant
?
Number of samples in 1 fuzzy dimension
S
10
between 2 and 32
The total number of samples is S^F + N*2^F, where S is the number specified here, F is the number of fuzzy variables, and N is the number of alpha levels, e.g. 10^2 + 10*2^2 = 140.
Build Kriging (Response Surface)
Yes - Kriging On (fast)
?
Number of points in response surface
K
5
between 2 and 50
The total number of points used to build the F-dimensional response surface is calculated using (F+K-1)!/(F!(K-1)!), where K is the number specified here, and F is the number of fuzzy variables, e.g. (2+5-1)!/(2!(5-1)!)=15. This formula is a heuristic rule that limits the maximum error when using more than one fuzzy variable.
Submit

Figure 5.2: Overview of the **Solver Settings**.

is 11, corresponding to the eleven equidistant alpha-levels `[0:0.1:1]`. The number of alpha-levels is a trade-off between accuracy and computational cost, although N is not necessarily an important factor in the total number of deterministic runs needed, as demonstrated in Section 5.4.2. Before the fuzzy analysis starts, all fuzzy input variables (see Section 4.3) are converted to N alpha-levels. Note: this alpha-level conversion is *not* applied to fuzzy numbers present in the calculator memory (workspace). To change the number of alpha-levels of fuzzy numbers stored in the workspace, use the `discretize` command of Section 2.3.

Select Optimisation Method : Click the grey bar to display a drop-down menu listing the various optimisation methods available. The default method is Sampling - Equidistant. Some theoretical background on the various methods is given further-on in Section 5.3. Click the question mark "(?)" hyperlink to the right of the grey bar

to access an on-line help page with a brief explanation of the methods. Alternatively, consult the Fuzzy Calculator User Guide in the fuzzy calculator tab. Depending on which method is selected, some of the input fields will automatically be enabled or disabled.

For Optimisation Method "Sampling - Equidistant" : This field is relevant only if the user has selected the method Sampling - Equidistant, see Section 5.3.2. For this method, the total number of samples is $S^F + N * 2^F$, where F is the number of fuzzy input variables, N is the number of alpha-levels and S is the number specified in the text field next to Number of samples in 1 fuzzy dimension. For example, for two fuzzy input variables $F = 2$, eleven alpha-levels $N = 11$ and ten samples per fuzzy dimension $S = 10$, the total number of samples is $10^2 + 11 * 2^2 = 144$.

For Optimisation Method "Sampling - Transformation Method" : This field is relevant only if the user has selected the method Sampling - Transformation Method, see Section 5.3.1. For this method, the total number of samples is $N * 2^F$, where F is the number of fuzzy input variables and N is the number of alpha-levels. For example, for two fuzzy input variables $F = 2$ and eleven alpha-levels $N = 11$, the total number of samples is $11 * 2^2 = 44$.

For Optimisation Method "GaD - Gradual Alpha-Level Decreasing" : This field is relevant only if the user has selected the method GaD - Gradual Alpha-Level Decreasing, see Section 5.3.3. This is a heuristic algorithm that decides on the spot whether to continue, split or abort a given search path, so it is not possible to give an estimate of the total number of samples needed. The user can specify the Aggression of the method, which is a positive integer that determines how quickly the algorithm will discard a search path. The default value is 2.

For all Optimisation Methods : Any optimisation method can be combined with Kriging. Kriging is one particular way to build a response surface. Once a response surface has been built, the samples are taken on the surface and not on the deterministic core itself. Turn Kriging on or off by clicking the grey bar next to Build Response Surface (Kriging)? By default, Kriging is on and a response surface is used. The use of Kriging will vastly accelerate the analysis, while the loss of accuracy is usually acceptable.

The number of points used to build the F-dimensional response surface is calculated using the formula $(F + K - 1)!/(F!(K - 1)!)$, where F is the number of fuzzy input variables and K is the number specified in the text field labeled Measure for number of points in response surface. K must be an integer between 2 and 50, default value is 5.

For example, for $F = 2$ fuzzy input variables and $K = 5$, the response surface is built on $(2+5-1)!/(2!(5-1)!) = 15$ points. This formula is a heuristic rule that limits the maximum error when using more than one fuzzy variable.

Note that F, the number of fuzzy input variables, only counts *truly* fuzzy inputs. When the user assigns a real value or a crisp membership function to a fuzzy input variable, then it does not count as a truly fuzzy input and it will not contribute to F.

5.3 Fuzzy Optimisation Methods

The following sections give a brief overview of the theoretical background of the various methods available in the **Solver Settings** box. For more details about these methods, please refer to the references mentioned in the text (and listed in the last section of this chapter) or to other specialized literature.

The Figures presented in the following sections assume that there are two fuzzy input variables, or $F = 2$. When we combine the stack of 1D intervals of the first fuzzy input (the alpha-levels of the membership function) with the stack of 1D intervals of the second fuzzy input, then we obtain a stack of 2D squares of various sizes, as shown in Figure 5.3. Also in 2D, the lower alpha-level squares contain the higher alpha-level squares (convexity of the membership function, Section 2.2).

5.3.1 Sampling - Transformation Method (TM)

In the Transformation Method (TM) [Hanss 2002] the function is evaluated, for every alpha-level, for all possible combinations of extremal values for the input parameters. In Design of Experiments terminology this is a so-called 2-level Full Factorial design. The parameter space is sampled on every vertex of the hyper-rectangles of the various alpha-levels, see Figure 5.3. The total number of samples is $N * 2^F$, where F is the number of fuzzy input variables and N is the number of alpha-levels. For example, for $N = 11$ and $F = 2$ fuzzy input variables, the fuzzy analysis will use $11 * 2^2 = 44$ samples. For higher values of F, the number of samples can be quite big and a response surface will be used to limit cpu time. Computationally, the TM scales exponentially with the number of fuzzy input parameters F, multiplied by the number of alpha-levels N, thus $O(N * 2^F)$. This method gives exact results only if the deterministic core function is **monotonic**, otherwise a severe underestimation of the output uncertainty may result. If the function is monotonic, then TM will give the same result as Equidistant Sampling and GaD.

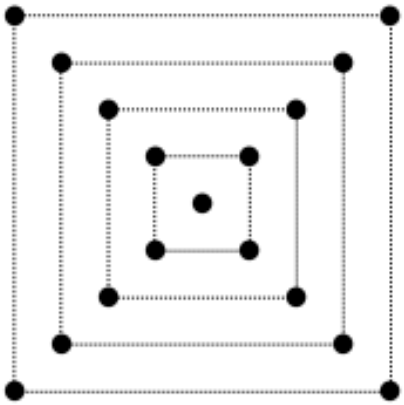

Figure 5.3: Transformation Method (TM): sampling the vertices only.

5.3.2 Sampling - Equidistant

The Sampling Equidistant method performs grid sampling over the entire hyper-rectangular parameter space. Every fuzzy input variable's support is sampled with an equal number of points S, to be specified on the **Solver Settings** box (default value is 10). In addition to this, the hyper-rectangular vertices are also used as samples.

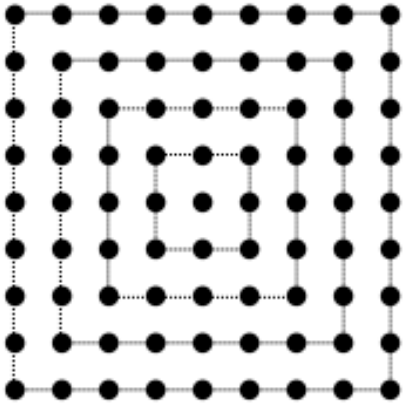

Figure 5.4: Sampling Equidistant: sampling vertices and internal values.

The total number of samples is $S^F + N * 2^F$, where F is the number of fuzzy input variables and N is the number of alpha-levels. For example, for $N = 11$, $S = 10$ and $F = 2$ fuzzy variables, the fuzzy analysis uses $10^2 + 11 * 2^2 = 144$ samples. Often this number is quite big and a response surface will be used to limit cpu time.

The Sampling Equidistant method thus scales exponentially with the number of fuzzy input parameters F, while it is almost independent of the number of alpha-levels N, see Section 5.4.2. The Sampling Equidistant method is exact only if the deterministic core function

is **monotonic**, otherwise an underestimation of the output uncertainty may result. However, since the parameter space is much more densely sampled than for the Transformation Method, the Sampling Equidistant method is expected to give a better estimate of the real output uncertainty.

5.3.3 Gradual alpha-level Decreasing (GaD)

Gradual alpha-level Decreasing [Degrauwe 2007] is a global optimisation approach. GaD initiates different search paths on the highest alpha-level (top) and prolongs them downward to the larger hyper-rectangular search spaces on lower alpha-levels, all the way down to the support, see Figure 5.5.

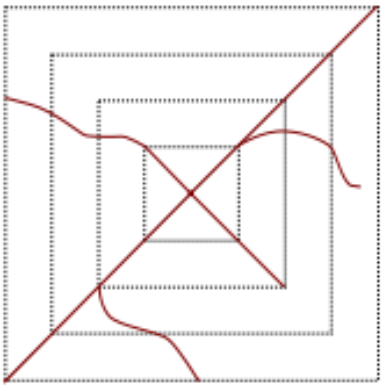

Figure 5.5: Gradual alpha-level Decreasing: from top (center) down to support.

Heuristics are used to decide whether a search path is interesting or not, and whether the path has to be terminated or not. New search paths can be created if it is believed there is an optimum between two existing search paths. Depending on the problem, the algorithm may perform very differently and may show very different computational complexity. If the problem is **monotonic**, then GaD will behave like the Short Transformation Method [Donders 2005] (currently not a part of the fuzzy webtools), which is an accelerated version of the Transformation Method and which is of linear complexity, $O(F)$. GaD will be slower when using more alpha-levels. GaD is more accurate than TM and Sampling Equidistant, and may be the fastest. However, one has to be careful with multiple output variables, as the GaD algorithm will try to optimise every output variable and they might not have their optimum in the same point, resulting in a large number of search paths. In this case, GaD may become too expensive and Sampling Equidistant becomes preferable.

Due to the heuristic character of the algorithm, the exact number of deterministic runs

required during the GaD scheme is not known in advance. The GaD library used by the Fuzzy website is the version of September 9, 2008.

5.3.4 Response Surfaces and Kriging

To find an optimum of a function (deterministic core) that is expensive to evaluate, it may be beneficial to construct a Response Surface, i.e. a surface that interpolates between some or all of the points that were already sampled (function evaluations already performed). The optimisation can then be done on the response surface itself, which is usually faster than working with the underlying function. The actual gain in speed depends on the nature of the underlying problem, the value of the input variables as well as the tuning of all available solver parameters. To guarantee that a local or global optimum has been found, one should apply a smart strategy of adding sample points for updating the response surface **during** the optimisation.

To avoid extraneous computational effort, a different approach is used here instead. The response surface is constructed fully **before** the optimisation starts. To improve the success and accuracy of the optimisation, an intelligent heuristic is used to select samples, aimed at improving the response surface in the neighborhood of possible optima. The downside of this approach is that some accuracy is lost as well as the guarantee of reaching a local or global optimum.

There exist various approaches to response surfaces. The most widely used approaches in engineering applications are: linear regression, radial basis functions and **Kriging**. On the fuzzy webtools website, the Kriging approach is used. Kriging considers the function as one particular realization of a stochastic process. By sampling this realization, one can construct a best linear unbiased estimator, which is then used as response surface. This approach is more than sufficient for many well-behaved functions, even if there is no stochastic process but just a deterministic function. For more information see for example [De Munck 2009].

On the **Solver Settings** pane, the user can specify a measure K for the number of points used to build the response surface. The total number of points used to build the F-dimensional response surface is calculated using $(F + K - 1)!/(F!(K - 1)!)$, where K is the measure specified on the Solver Settings box, and F is the number of fuzzy input variables. For example, for $F = 2$ fuzzy input variables and $K = 5$, the response surface contains $(2 + 5 - 1)!/(2!(5 - 1)!) = 15$ points. This formula is a heuristic rule that limits the maximum error when using more than one fuzzy variable.

Advantages Of Kriging

- Kriging is an exact interpolator (if the control point coincides with a grid node).
- Kriging variance:
  - relative index of the reliability of estimation in different regions,
  - good indicator of data geometry,
  - smaller nugget (or sill) gives a smaller Kriging variance,
  - minimizes the Mean Square Error,
  - can use a spatial model to control the interpolation process,
  - a robust technique, i.e. small changes in Kriging parameters equals small changes in the results.

Disadvantages Of Kriging

Kriging tends to produce smooth images of reality, as do all interpolation techniques. In doing so, short scale variability is poorly reproduced, while it underestimates extremes (high or low values). It also requires the specification of a spatial covariance model, which may be difficult to infer from sparse data. Kriging consumes much more computing time than conventional gridding techniques, requiring numerous simultaneous equations to be solved for each grid node estimated. The preliminary processes of generating variograms and designing search neighborhoods in support of the Kriging effort also requires much effort. Therefore, Kriging is probably not normally performed on a routine basis. It is rather best used on projects that can justify the need for the highest quality estimate of a structural surface and which are supported by plenty of good data.

5.4 Solver Settings and CPU time

5.4.1 The "Estimate CPU time" Button

As discussed in Section 4.5, the user must click the Estimate CPU time button before the calculation of the problem can be started. This is done in order to inform the user of the expected total CPU time needed, and to warn the user in case the chosen solver settings are such that they lead to an excessively large CPU time. The total CPU time needed is roughly proportional to the total number of samples required to complete the fuzzy analysis. The total number of samples required by various methods has been discussed in detail in previous sections.

After clicking the Estimate CPU time button, several lines of information appear, for example as in Figure 5.6:

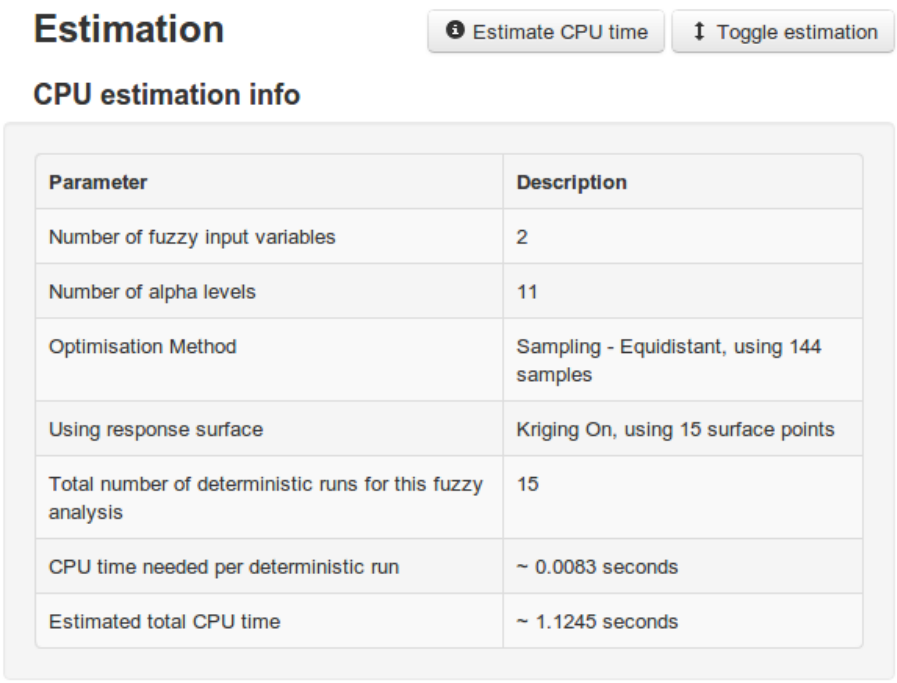

Figure 5.6: Information given after clicking the Estimate CPU time button.

- Number of fuzzy input variables : The number of fuzzy input variables F for the selected example problem. Fuzzy input variables have a green button next to their text field in the Input variables area. Note that F will become smaller when a real value or a crisp membership function is assigned to a fuzzy input variable.
- Number of alpha levels : The number of alpha-levels N.
- Optimisation Method : The selected optimisation method, and the number of samples it requires.
- Using response surface : Indicates whether Kriging is turned on (default) or turned off. If Kriging is turned on (i.e. using a response surface) then the number of points used to build the response surface is given.
- Total number of deterministic runs: The number of times the deterministic core module will be run in order to complete the fuzzy analysis. With Kriging turned on, this is the number of points used to build the response surface. With Kriging turned off, this is the number of samples required by the selected method.
- CPU time needed per deterministic run: CPU time needed for one single run of the deterministic core module of the selected problem.

- Estimated total CPU time: The product of the previous two lines.

Most example problems finish in one minute or less when using the default solver settings. If the last line Estimated total CPU time indicates more than 30 minutes then the user might consider selecting different solver settings and/or simplifying the problem. If the user intends to wait for the results of the fuzzy analysis, then the estimated total CPU time is a good indication of how much time is needed to complete the fuzzy analysis. Do not close the browser while a fuzzy analysis is running, or the results will be lost.

5.4.2 Effect of Parameter Values and Method on CPU time

The optimisation method and solver settings chosen by the user determine the accuracy of the solution (output variables) as well as the total CPU time required for the fuzzy analysis, which is roughly proportional to the number of deterministic runs required for the fuzzy analysis, see for example the formulas presented in Section 5.2 and repeated in Section 5.3.

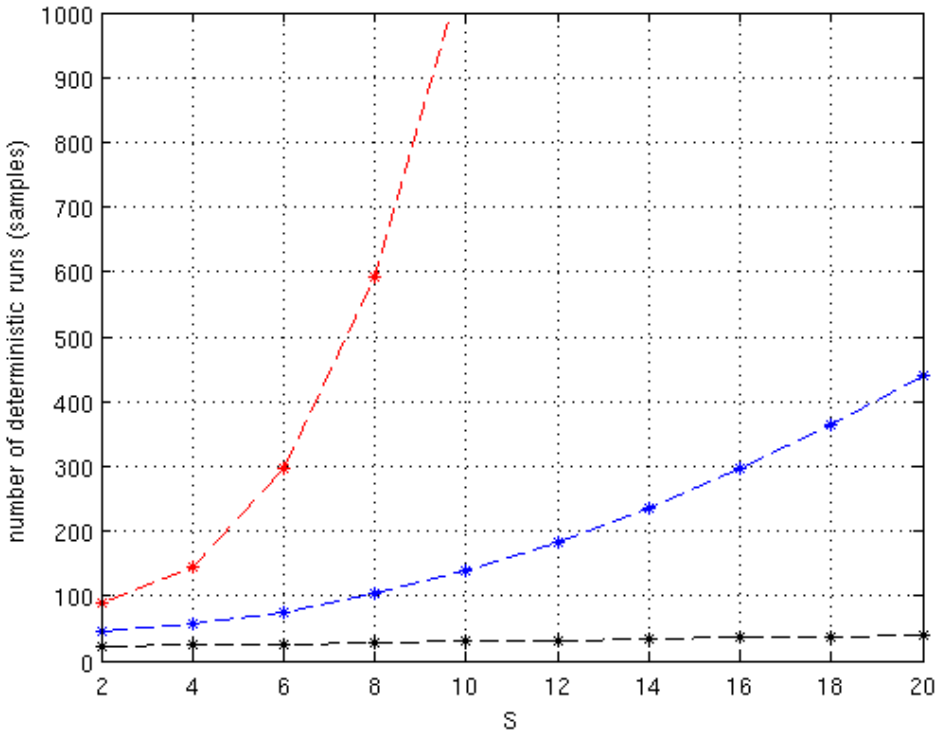

Figure 5.7: Number of deterministic runs required for the fuzzy analysis, as a function of S and F, for the Sampling - Equidistant method without Kriging: $F = 1$ (black, under), $F = 2$ (blue, middle), $F = 3$ (red, top).

For example, for the Sampling - Equidistant method without Kriging, the number of

deterministic runs is given by the formula $S^F + N * 2^F$, where F is the number of fuzzy input variables, N is the number of alpha-levels and S is the number specified on the **Solver Settings** box in the text field labeled Number of samples in 1 fuzzy dimension. For example, for two fuzzy input variables $F = 2$, eleven alpha-levels $N = 11$ and a sampling parameter value $S = 10$, the total number of deterministic runs is $10^2 + 11 * 2^2 = 144$. If the user prefers a greater sampling density, then the user must assign a higher value for S on the **Solver Settings** box. For example, setting $S = 20$ will perform a fuzzy analysis using $20^2 + 11 * 2^2 = 444$ deterministic runs for the same values $F = 2$ and $N = 11$. Using this formula the user can plot the number of deterministic runs for various values of S or N or F.

For the above example, Figure 5.7 shows the number of deterministic runs required as a function of S and F. The number of deterministic runs required for the fuzzy analysis scales exponentially with increasing number of fuzzy input variables F, because F determines the dimensionality of the search space. Keep the dimensionality F as small as possible, e.g. by assigning real values or crisp membership functions to fuzzy input variables that are currently of no interest.

Note that the number of alpha-levels N can also be seen as a sampling density in the interval [0,1] that covers the membership function image.

Example 1 – We first compare the Sampling - Equidistant method without Kriging. The parameter values on the Solver Settings box are set to $N = 11$ alpha-levels and $S = 10$ samples per fuzzy dimension. The number of fuzzy input variables is determined by the example problem. For the BVP with fuzzy parameter (2D) problem we have $F = 2$. After pressing the Estimate CPU time button, a summary of the selected method and solver settings is presented, and the Calculate! button is enabled. When the problem has been solved, the actual CPU time that was used is stored in the real-valued output variable `cputime` whose value is also immediately visible under the Problem Output section.

The resulting (average) CPU times are summarized in Table 5.1 for four combinations of N and S. The second column indicates the number of samples (deterministic runs) required for the fuzzy analysis, following the formula $S^F + N * 2^F$ with $F = 2$. We see that indeed the total CPU time is roughly proportional to the total number of samples.

Example 2 – We now fix the number of alpha-levels to $N = 11$ and compare the cases $F = 2$ and $F = 1$. We arrive at one fuzzy input variable $F = 1$ by defining one of the fuzzy input variables of the example problem by a real value or a crisp membership function. We

N, S	# samples	total CPU time (s)	CPU time/sample (s)
$N = 11, S = 10$	144	27.1	0.188
$N = 11, S = 20$	444	84.7	0.191
$N = 22, S = 10$	188	33.4	0.178
$N = 22, S = 20$	488	89.2	0.183

Table 5.1: Number of samples and actual CPU times for various combinations of solver settings for the Sampling - Equidistant method without Kriging, applied to the BVP with fuzzy parameter (2D) example problem ($F = 2$).

arrive at $F = 0$ by defining both fuzzy input variables by a real value or a crisp membership function. Table 5.2 shows that indeed the number of samples required by the fuzzy analysis increases dramatically with F, and that the total CPU time is roughly proportional to the total number of samples.

F, S	# samples	CPU time (s)	CPU time/sample (s)
$F = 0$	1	0.31	0.310
$F = 1, S = 10$	32	5.41	0.169
$F = 1, S = 20$	42	7.64	0.182
$F = 2, S = 10$	144	27.1	0.188
$F = 2, S = 20$	444	84.7	0.191

Table 5.2: Number of samples and actual CPU times for various combinations of solver settings for the Sampling - Equidistant method without Kriging, applied to the BVP with fuzzy parameter (2D) example problem: comparing $F = 1$ and 2 ($N = 11$).

Example 3 – Next we look at the effect of Kriging: we compare the Sampling - Equidistant method without and with Kriging. We can switch Kriging on or off on the **Solver Settings** tab, and specify the measure K in the text field labeled Measure for number of points in response surface. K determines the number of points (deterministic runs) used to build the response surface, according to the formula $(F+K-1)!/(F!(K-1)!)$ where F is the number of fuzzy input variables. For our BVP with fuzzy parameter (2D) example problem, we have $F = 2$. For $K = 5$ the response surface is thus built using 15 points. For $K = 10$ the response surface is built using 55 points.

The resulting (average) CPU times are summarized in Table 5.3 for the analysis without Kriging, with Kriging $K = 5$ and with Kriging $K = 10$. The second column indicates the number of samples required by the Sampling - Equidistant method for sampling densities $S = 10$ and $S = 20$. Note that these samples are now taken on the response surface, and do not require a deterministic run of their own! The third column indicates the number

of points used to build the response surface: every point requires a deterministic run. The last column shows that it is not the number of samples (not the sampling density) that determines the total CPU time required for the analysis, but rather the number of points, i.e. the number of deterministic runs.

Settings	# samples	# points	CPU time (s)	CPU time/point (s)
$S = 10$, Kriging Off	144	–	27.1	–
$S = 20$, Kriging Off	444	–	84.7	–
$S = 10$, Kriging On $K = 5$	144	15	3.50	0.233
$S = 20$, Kriging On $K = 5$	444	15	3.74	0.249
$S = 10$, Kriging On $K = 10$	144	55	11.1	0.202
$S = 20$, Kriging On $K = 10$	444	55	11.2	0.204

Table 5.3: Number of samples, number of response surface points and actual CPU times for various combinations of solver settings for the Sampling - Equidistant method *without and with* Kriging, applied to the BVP with fuzzy parameter (2D) example problem ($F = 2, N = 11$).

Settings	# points	total CPU time (s)
GaD, Kriging Off	–	86.9
GaD, Kriging On $K = 5$	15	90.3
GaD, Kriging On $K = 10$	55	22.6
GaD, Kriging On $K = 15$	120	27.2

Table 5.4: Actual CPU times for various combinations of solver settings for the GaD - Gradual Alpha-Level Decreasing method *without and with* Kriging, applied to the BVP with fuzzy parameter (2D) example problem ($F = 2, N = 11$, Aggression=2).

Example 4 – We now look again at the effect of Kriging, but for a different method: we compare the GaD - Gradual Alpha-Level Decreasing method without and with Kriging. Please refer to Example 3 above with respect to the Kriging settings. On the **Solver Settings** tab, select the GaD method and specify a positive integer in the text field labeled Aggression, or keep the default value 2.

The resulting (average) CPU times are summarized in Table 5.4. Note that the number of samples used by the GaD method is unknown. In case Kriging is turned on, the number of points used to build the response surface determines the number of deterministic runs required. We see that the use of a response surface will not accelerate the GaD method unless if the surface is sufficiently detailed: only for $K \geq 10$ ($\geq$55 points) there is an advantage of using Kriging.

5.5 References

Hanss Michael, *The transformation method for the simulation and analysis of systems with uncertain parameters*, Fuzzy Sets Syst. **2002**, vol 130(3) pp.277-289, Elsevier North-Holland Inc., Amsterdam, The Netherlands.

Donders S., Vandepitte D., Van de Peer J. and Desmet W., *Assessment of uncertainty on structural dynamic responses with the short transformation method*, Journal of Sound and Vibration **2005**, vol 288(3) pp.523-549.

Degrauwe Daan, *Uncertainty propagation in structural analysis by fuzzy numbers*, Ph.D. thesis K.U.Leuven, Belgium, **2007** (download PDF file here).

De Munck M., *Efficient Optimisation Approaches for Interval and Fuzzy Finite Element Analysis*, Ph.D. thesis K.U.Leuven, Belgium, **2009** (download PDF file here).